

Remarks

Applicants respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 1-36 are pending in the application. Claims 1-36 are rejected. No claims have been allowed. Claims 1, 21, 25, 27, 29, and 32 are independent. Claims 1, 15, 18, 21, 25, 27, 29, and 32 have been amended.

Cited Art

The Action cites:

U.S. Patent Application No. 11/330,053 (Pub. No. 2006/0129994) to Srivastava et al., entitled "Method and Apparatus for Prioritizing Software Tests" ("the '053 application");

Srivastava et al., "Effectively Prioritizing Tests in Development Environment," February 2002, MSR-TR-2002-15, Publisher: *Association for Computing Machinery, Inc.* ("Srivastava 2"); and

Srivastava et al., "Vulcan: Binary transformation in a distributed environment," April 2001, Technical Report MSR-TR-2001-50 (Srivastava).

Claim Objection

The Action objects to claim 18 for an informality. Applicants note that the claim language reciting "logical abstractions effected by the proposed change" was a typographical error and thank the Examiner for the opportunity to correct it. The claim has been amended to recite "wherein the metric for proposed change risk comprises a number of logical abstractions *affected* by the proposed change." [Emphasis added.] Applicants believe that the amendment should obviate the objection.

Claim Rejections - 35 USC § 112

The Action rejects Claims 1-20, and 27-28 under 35 USC § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Specifically, the Action notes insufficient antecedent basis for the language “receiving, responsive to the requests” as recited in claims 1 and 27. Applicants note that this was a typographical error and thank the Examiner for the opportunity to correct it. Both claims have been amended to recite “receiving, responsive to the request,” which has antecedent basis in each claim. Applicants believe that claims 1 and 27, as well as dependent claims 2-19 and 28 are now patentable under 35 U.S.C. § 112 and respectfully request that the rejections under § 112 be withdrawn.

Claim Rejections under 35 USC § 102 over Srivastava

The Action rejects claims 1-16, 20, 27-28, 32, and 36 under 35 USC 102(b) as being anticipated by Srivastava. Applicants respectfully submit the claims are allowable over the cited art. For a 102(b) rejection to be proper, the cited art must show each and every element as set forth in a claim. (See MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn. Claims 1, 27, and 32 are independent.

Claim 1

Claim 1, as amended, recites, in part:

requesting, via an application programming interface, a dependency collection, by sending to the application programming interface a system dependency creation request comprising the received system definition and a dependency request comprising a target logical abstraction, *the target logical abstraction indicating a chosen structural level of complexity of a system described by the system definition;*

receiving, responsive to the requests sent via the application programming interface, a dependency collection for the target logical abstraction comprising logical abstractions in one or more dependency chains with the target logical abstraction, wherein the dependency collection comprises logical abstractions outside of a subsystem for the target logical abstraction and *the dependency chains represent chained dependencies between subsystems; . . .*

[Emphasis added.] For example, the Application describes examples of logical abstractions:

In this implementation, a binary file has code groupings within a binary file (e.g., basic blocks, functions, procedures, objects, and or other logical abstractions).

[Application, at page 17, lines 12-13.] The application goes on to describe the utility of allowing various levels of logical abstraction, as well as chains of dependencies:

Using the described interface 1200, a tool 214-218 is programmed presenting a GUI that exposes for example, what binary files outside a binary file's subsystem, depends on a binary. *Further, the methods allow the tool to drill down further into what procedures, functions, or even basic blocks, call a procedure, function, or basic block from anywhere in the system. By iterating through the dependency graph, a logical abstraction is selected (e.g., node, basic block, procedure, etc.), and the logical abstractions that depend directly or indirectly on that logical abstraction, can be identified.*

[Application, at page 18, lines 1-8; emphasis added.] The Application then continues, describing how dependencies can chain between subsystems:

For example, a first logical abstraction in a first binary in a first subsystem, is exposed as having hundreds or thousands of direct or indirect dependencies, whether inside or outside the first logical abstraction, inside or outside the first binary, or inside or outside the first subsystem. *Even chains of dependencies running in and out of multiple subsystems are discoverable and exposable with the described variations of technologies.* Even before a binary file is changed, a system is defined and discovered, and the risks associated with a proposed change within a logical abstraction can be evaluated.

[Application, at page 18, lines 8-15; emphasis added.]

The input abstraction of Srivastava does not teach or suggest a "a system dependency creation request" which comprises a "the target logical abstraction [which] indicat[es] a chosen structural level of complexity of a system described by the system definition" as recited in claim 1. In its rejection of the above-quoted language of claim 1, the Action cites to § 2.1, page 5 of Srivastava, finding "input form of abstraction submitted in an API to yield output level via externalizing . . . the received abstractions." [Action, at page 6.] Applicants note, however that while § 2.1 of Srivastava does show an "abstract representation" of a program, there is no discussion that the representation indicates any particular level of complexity of a system. In fact, Applicants respectfully note that an abstract representation *of an entire program*, such as is shown in Srivastava, would be focused on representing the program itself, and would have no purpose as an indicator of a "chosen structural level of complexity of a system," as any abstraction would have already taken place by the time the representation was made. Additionally, as Figure 3 on page 5 shows, various different abstractions may be contained in a

single Vulcan program representation. This means the representation itself would have no use as an indicator, since may contain various, competing, levels of abstraction.

Furthermore, Applicants note that there is no indication that the “abstract representation” of § 2.1 is included in any requests. In fact, Figure 2 of Srivastava appears to indicate that the illustrated APIs directly operate *on* the abstract representation, rather than through requests which include it. For at least these reasons, Srivastava does not teach or suggest the above-quoted language of claim 1.

Neither the “branch chaining” example nor the “program representations” example of Srivastava teach or suggest a “dependency chains represent[ing] chained dependencies between subsystems” as recited in claim 1. In its rejection of the “dependency chain” language of claim 1, the Action cited to § 2.2, page 5 of Srivastava, as well as Appendix A. Applicants respectfully note that neither of these sections teaches or suggests “dependency chains represent[ing] chained dependencies between subsystems.” Section 2.2 on page 5 illustrates the general structure of “Vulcan program representations.” While the Figure does illustrate “six basic abstractions” which are used to represent programs abstractly in Vulcan, there are no dependencies illustrated between the abstractions, nor do the Applicants find any such discussion in § 2.2.

Furthermore, the example given in Appendix A, which is of “a peephole optimization that performs branch chaining” does not read on the above-quoted language either. [Srivastava, at § 2.2, page 5.] The branch chaining optimization in Appendix A has the purpose of chaining together program components that are connected by unconditional branches. [Srivastava, at Appendix A, page 11.] Its final output is written source code with chained branches. [*Id.*] As such, it does not teach or suggest a representation of any sort of “chained dependenc[y]” because it creates an actual program component not a dependency representation. Additionally, by performing an optimization which specifically chains program parts, the optimization of Appendix A will necessarily remove or change any existing dependencies. Thus, it cannot represent dependencies between subsystems when it is modifying them as well.

Thus, Srivastava does not teach or suggest at least the above-quoted language of claim 1. Applicants therefore request that the rejection of claim 1, as well as its dependent claims 2-16 and 20 be withdrawn and that the claims be allowed.

Claim 27

Claim 27, as amended, recites, in part:

the target logical abstraction indicating a chosen structural level of complexity of a system described by the system definition;
[]the dependency chains represent chained dependencies between subsystems

For at least the reasons given above with respect to claim 1, Srivastava does not describe at least the above-quoted language of claim 27. Srivastava thus does not read on every limitation of claim 27. Applicants therefore request that the rejection of claim 27, as well as its dependent claim 28, be withdrawn and that the claims be allowed.

Claim 32

Claim 32, as amended, recites, in part:

means for accepting an indication of a target logical abstraction indicating a chosen structural level of complexity of the system; and
means for displaying dependency relationships between the target logical abstraction and a set of logical abstractions in binary files between two or more of the plural subsystems.

For at least the reasons given above with respect to claim 1, Srivastava does not describe at least the above-quoted language of claim 32. Srivastava thus does not read on every limitation of claim 32. Applicants therefore request that the rejection of claim 32, as well as its dependent claim 36, be withdrawn and that the claims be allowed.

Claim Rejections under 35 USC § 102 over Srivastava 2

The Action rejects claims 21-26 and 29-31 under 35 USC 102(b) as being anticipated by Srivastava 2. Applicants respectfully submit the claims are allowable over the cited art. For a 102(b) rejection to be proper, the cited art must show each and every element as set forth in a claim. (See MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn. Claims 21 and 29 are independent.

Claim 21

Claim 21, as amended, recites, in part:

determining *dependency information about binary files comprising entry and exit points* for the binary files;
propagating dependency information about binary files to determine *subsystem dependency information comprising entry and exit points for a subsystem* of a system;
propagating the subsystem dependency information to determine *system dependency information comprising entry and exit points for the system*;
marking changed logical abstractions;
marking unchanged logical abstractions dependent on marked changed logical abstractions in other subsystems;

[Emphasis added.] The Application discusses an example process of the propagation recited in claim 21 with reference to Figure 13:

Figure 13 is a flow chart 1300 of an exemplary method for marking basic blocks that are new or changed with respect to a previous version, and for marking basic blocks that are unchanged if they depend directly or indirectly on changed basic blocks.

At 1302, the method receives or defines a system definition (e.g., a system definition file).

At 1304, the method determines for each binary file in the system, information about entry and exit points, and stores the information in a record associated with the binary file (e.g., Figure 5, at 504).

At 1306, the method determines entry and exit points for each subsystem within the system, and for the system (e.g., Figure 5, at 506).

[Application, at page 19, lines 4-13; emphasis added.]

The propagation of changes found in binary files used in Srivastava 2 does not "propagate[e] dependency information" where "dependency information compris[es] entry and exit points" as recited in claim 21. In its rejection of claim 21, the Action cites to the 5th paragraph of the right column of page 2. The particular cited sentence states:

Moreover, by the time the program is available in binary form, all the changes in header files to constants, macro definitions, etc. have already been propagated to the affected procedures in the program, thus simplifying the process for determining program changes.

As in a previous amendment, Applicants note that that which is being propagated in the cited passage are *changes*, not dependencies. Thus, notwithstanding Applicants contention that no propagation is taught by Srivastava, even if the passage were to be interpreted to include

propagation, there is no indication given in Srivastava of the propagation of any dependency information which comprises entry and exit points. Applicants note that this applies not only to the binary file-level dependency information, but to the subsystem and system dependency information of claim 21 as well. Applicants also respectfully submit that the amendments are responsive to the discussion of pages 17 and 18 in the Response to Arguments section of the Action, where it is suggested that the actions of determining and propagating need be more clearly conveyed.

The impacted and matching blocks of Srivastava 2 do not teach or suggest “marking unchanged logical abstractions dependent on marked changed logical abstractions in other subsystems” because there is no indication in Srivastava of using dependencies to mark blocks. The Application describes examples of “marking unchanged logical abstractions dependent on marked changed logical abstractions”:

At 1310, the method propagates the changes to compute the affected parts of the system by performing analysis at each of three levels of abstractions—binary, subsystem, and system. *For example, as discussed in view of Figure 14, the propagation determines what basic blocks depend on the marked basic block. The blocks that depend directly or indirectly on a marked (affected) basic block are marked during propagation.*

[Application, at page 20, lines 13-18; emphasis added.] The Application also describes the motivation for marking such unchanged blocks:

This information (marked blocks) is used, for example, to determine how an affected basic block might affect an unchanged basic block in another subsystem. In one case, this information is used to exercise tests that execute unchanged basis blocks that depend on affected blocks elsewhere in the system.

[Application, at page 20, lines 18-21; emphasis added.]

The passage cited in the Action against the above-quoted language of claim 21 discusses “using BMAT to find [] matching block[s] in the old binary for each new block in the new binary.” [Srivastava 2, at page 3, right column.] The passage discusses “new blocks” which are blocks with no matches, old blocks, which are identical, and old modified blocks, which have matches but are not identical. The passage then goes on to determine “which impacted blocks,” i.e. which old modified block or new blocks, “are likely to be covered by an existing test.” [*Id.*]

Thus, in contrast to the claim language and the examples shown above, nowhere in this passage does Srivastava 2 discuss utilizing dependencies between unchanged and changed blocks (let alone “logical abstractions” as recited in the claim). Instead, blocks are considered

solely on whether they match or are identical between versions; dependency information is not used. Among other effects, this means the benefits discussed in the passage from the Application quoted above cannot be realized by Srivastava 2 because no dependency information is utilized.

Additionally, Srivastava 2's determination of test coverage also could not teach or suggest this claim language. In Srivastava 2, unchanged blocks, even if dependent on changed blocks, would be considered "old" and thus not considered when Srivastava determines which blocks are covered by tests. Thus the process could not read on "marking" these blocks.

For at least these reasons, Srivastava 2 does not teach or suggest at least the above-quoted language of claim 21. Applicants therefore request that the rejection of claim 1, as well as its dependent claims 22-24, be withdrawn and that the claims be allowed.

Claim 25

Claim 25, as amended, recites, in part:

- means for determining binary dependencies, comprising entry and exist points, for a defined system;
- means for propagating binary dependencies to identify binaries dependent on binaries in other subsystems;
- ...
- means for propagating marked changes using the determined and propagated dependencies; and

For at least the reasons given above with respect to claim 21, Srivastava 2 does not describe at least the above-quoted language of claim 25. Srivastava 2 thus does not read on every limitation of claim 25. Applicants therefore request that the rejection of claim 25, as well as its dependent claim 26, be withdrawn and that the claims be allowed.

Claim 29

Claim 29, as amended, recites, in part:

- determining dependency information for binary files comprising entry and exit points for the binary files;
- propagating dependency information about binary files to determine subsystem dependency information comprising entry and exit points for a subsystem of a system; and

propagating subsystem dependency information to determine system dependency information comprising entry and exit points for the system;

... propagating marked changes according to the dependency information comprising marking unchanged logical abstractions dependent on marked changes in other subsystems;

For at least the reasons given above with respect to claim 21, Srivastava 2 does not describe at least the above-quoted language of claim 29. Srivastava 2 thus does not read on every limitation of claim 29. Applicants therefore request that the rejection of claim 29, as well as its dependent claim 2 30 and 31, be withdrawn and that the claims be allowed.

Patentability of Claims 17-19 and 33-35 under 35 USC § 103(a)

The Action rejects claims 17-19 and 33-35 under 35 U.S.C. § 103(a) as unpatentable over Srivastava in view of Srivastava 2. Applicants respectfully submit the claims in their present form are allowable over the cited art. To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. (MPEP § 2142.)

For at least the reasons discussed above, Srivastava does not teach or suggest at least one limitation of claims 17-19 and 33-35, which depend from claims 1 and 32 respectively. Furthermore, applicants do not find relevant disclosure in Srivastava 2. Applicants thus respectfully request that the rejection of claims 17-19 and 33-35 be withdrawn and that the claims be allowed.

Double Patenting Rejection

The Action maintains its provision rejection of claims 21, 25, and 29 under the judicially created doctrine of obviousness-type double patenting as being unpatentable over the '053 application in view of Srivastava. Applicants respectfully submit the claims in their present form are patently distinct over the cited art.

In the course of the provisional double patenting rejection, the Action admits that '053 application claims do not recite a number of features, including:

determining dependency information about binary files, propagating such information to determine subsystem and system dependency, marking changed and unchanged logical abstraction to prioritize tests.

[Action, at page 3.] Instead, the Action finds these features in Srivastava.

Applicants respectfully note that, for at least the reasons given above with respect to claims 21, 25, and 29, Srivastava does not teach or suggest at least one of the features of each claim. For example, Srivastava does not teach or suggest "propagating dependency information about binary files to determine subsystem dependency information comprising entry and exit points for a subsystem of a system" and "propagating the subsystem dependency information to determine system dependency information comprising entry and exit points for the system" as recited in claim 21. Because language from each of claims 21, 25, and 29 is found neither in the '053 application claims nor in Srivastava, such language is not taught or suggested by the combination of the two.

For at least this reason, the Action fails to make a establish a *prima facie* case of obviousness over the '053 Application in view of Srivastava. Accordingly, applicants request that the provisional double patenting rejection be withdrawn.

Interview Request

If the claims are not found by the Examiner to be allowable, the Examiner is requested to call the undersigned attorney to set up an interview to discuss this application.

Conclusion

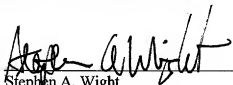
The claims in their present form should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By


Stephen A. Wight
Registration No. 37,759